Report No. 346

COO-1469-0144

GRAPHICAL REMOTE-ACCESS SIMULATION SYSTEM (GRASS)
The Communication and Monitor Components (GASP/GLASP)

by

Martin Joel Michel

August 1969

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 346

GRAPHICAL REMOTE-ACCESS SIMULATION SYSTEM (GRASS)
The Communication and Monitor Components (GASP/GLASP)*

by

Martin Joel Michel

August 1969

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This work was supported in part by Contract U. S. AEC AT(11-1)1469 and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, January 1970.

ACKNOWLEDGMENT

The author is indebted to Dr. C. W. Gear whose suggestions and comments helped greatly during the development and writing of this thesis.

PREFACE

This paper describes a communications component for an online, graphical, simulation system operating in a multiterminal environment. The desired features, hardware/software tradeoffs, and other design considerations of the overall system are briefly discussed followed by a description of the communications package. The current implementation level, problems, and possible developmental trends are also described.

TABLE OF CONTENTS

LIST OF FIGURES

I. THE CURRENT ENVIRONMENT

The utility and advantages of interactive online graphics terminals have been well demonstrated over the past several years.[1] The effectiveness of simulation studies is obvious.[2] A marriage of the two is also clear and has already been performed many times.[3] However, powerful graphical simulation systems are not at all common, and are in fact only in regular use at a small number of large commercial companies and government agencies. Two primary reasons for this result are high cost and low flexibility.

Typically, a large CPU with extensive peripherals is used to service at most two or three directly attached graphics terminals. Often the same CPU is doing some batch processing in background while also running a timesharing service at standard keyboard terminals. Such a configuration tends to reduce the superficial cost of the graphics support, but results in an unpleasant degradation in system performance. Discrete response time at the graphics terminals is long. Certain resources, such as CPU core allocated to the graphics terminals, remain idle much of the time and so reduce batch and timesharing capabilities; and certain often trivial graphics operations, such as pentracking, can completely saturate the CPU. On the other hand, to dedicate the CPU to graphics support quickly increases the cost.

Recently, a compromise of sorts has come into increasing prominence. A small, or relatively small, general purpose CPU (8 to 32 k of core, $15,000 to $100,000 price range) is used to perform the local functions (such as display manipulation, data structure creation, parameter assignment, prompting, etc.) needed to effectively

interact with a user at a display console ($30,000 to $90,000 price range) in a relatively fast response environment.[4] This small CPU is attached in satellite mode to a large central CPU which provides library, documentation, and particularly number-crunching analysis facilities. Hence, after specifying his problem, the console-user theoretically has to endure a long response time only when analysis is being performed in the central CPU. However, "long" in this case is only the analysis time, not analysis time, plus card punching time, plus turn-in time, plus backlog time, plus print time, plus return time.

Yet many problems still persist. 1) The satellite CPU is typically capable of only supporting one display console, so per-terminal investment is still not small. 2) The satellite CPU is incapable of some forms of desirable operations (such as picture rotations). 3) The satellite CPU has very limited data storage capabilities, thus limiting picture complexity and, more important, severely limiting the amount of necessary library information that can be stored locally. Hence, access to the central CPU library becomes very frequent as the number of users and the library grow. This lengthens, in turn, response time for functions (such as picture-sub-picture generation) that are considered "local functions." 4) The speed of the data linkage between the satellite and central CPU's is a critical factor in the amount and type of data that can be passed back and forth. This affects the division of labor between the CPU's and the consequent response time at the display console. 5) Resources in the central CPU are still wasted, since allocation of resources

must be made at the beginning of a session, even though actual analysis and maximum utilization may not occur until the end.

The purpose of a graphical simulation system is to aid a user in defining, analyzing, and solving his particular problem. Certain terms just mentioned must be stressed: "aid" and "his particular problem." First, the system must strive to avoid hindering the user or complicating his problem solving attempts with system-dependent details and restrictions. This is, of course, an ideal in a world where most users are people not intimately familiar with computers, but who want to use them in pursuing their other work. By necessity, any system must have certain formalisms and conventions, but these should, and must, be implemented such that they are natural and convenient to the neophyte, as well as the experienced, user. Presently, most systems are burdensome and lack adequate prompting and self teaching facilities.

Second, the system should attempt to solve as wide a range of problems as possible. This is easy to say--it is what most people usually try to accomplish--but it is a goal quite difficult to attain. Generality usually means obtaining results for a specific problem in more time than would be needed by a special purpose approach. Hence, in designing a system, the class of problems to be solved is often chosen so that the differences from one problem to another will cause a relatively insignificant change in approach, and thus, in system performance. Unfortunately, when the system is implemented, certain system characteristics (core space, data structure, etc.) usually decrease the acceptable problem class. In addition, implementation

of the analysis proceedures usually makes use of many element dependent characteristics, such as in electrical network analysis programs. That is, an electrical network analyzer cannot do much with a chemical cracking plant piping network, although both are topologically similar. Naturally, the class of acceptable problems is further reduced; the user's "particular problem" becomes very particular indeed. Presently, most systems support special purpose graphics interfaces to at most a few special purpose analysis packages.

FOOTNOTES

[1] Beginning with Sutherland's classic "SKETCHPAD" in 1963, graphics terminals and their applications as flexible man-machine interfaces have been speedily proliferating. Interested readers should scan such publications as Datamation, Computers and Automation, Information Display, and IEEE Spectrum to get an idea of the hardware/software commercially available. For a state-of-the-art view, a good starting point would be AFIPS, and ACM Conference Proceedings as well as proceedings from special interest conferences, such as the recent "Pertinent Concepts in Computer Graphics," University of Illinois, March 1969. A historical perspective of graphics in the computer field can be obtained from articles such as the one by Hobbs.

[2] The adage "everybody's doing it" applies directly; almost every technical journal in any field contains articles about simulation of phenomena peculiar to that field. Areas of application range from the obvious (aircraft design and circuit analysis), to the more obvious (business management and stock market trends). Places to look besides specific professional journals include the communications of the ACM, Simulation, and IEEE Proceedings. General articles include those by Harmon, Blake, and Shigley.

[3] Typical systems include aircraft design (Lockheed), automotive design (General Motors), NASA space simulation (General Electric), electrical circuit analysis (ECAP variations), IC design (MIT), ship building design (CDC), chemical cracking networks (Brown), ad infinitum. Simply look for simulation studies; a graphical implementation is often included.

[4] The DEC 338 and IBM 1130-MOD IV are typical examples of the low and high ranges, respectively, of such systems.

II. A DESIRED ENVIRONMENT

The goal in designing a graphical simulation system, as indicated in the previous section is 1) to use a hardware configuration that reduces the cost per terminal while maintaining the capabilities and response of a dedicated system, and 2) to enlarge the class of acceptable problems that the system can deal with. This is just the universal "more product for less investment" goal; the question is the usual--HOW?

Until the time that a small CPU and display console can be built economically as a single unit to be attached directly in time-sharing mode to a central CPU, the use of an intermediate satellite CPU will be necessary. However, as previously discussed, several deficiencies exist in present graphical console, satellite CPU, central CPU configurations. To alleviate these, the following provisions should be made:

1. The satellite drives a multiple number of consoles.
2. The satellite has a local mass storage facility.
3. A high speed data link exists between the two CPU's.

Supporting many terminals with one satellite will drive down the average cost. Local mass storage reduces the need to access the central library. A high speed data link makes central library accessing and other necessary data transfers low in cost with respect to time.

Two main problems in supporting a multiconsole environment are initial picture generation and picture refreshing for each individual console. During picture generation, a large amount of data manipulation is necessary, along with a high frequency of library accessing. Hence,

if the satellite is busy generating a picture for one console and a second console requests service, the response time at the second console will be greatly delayed. Moreover, all the data structures associated with the current pictures on the consoles will not fit in the satellite's core simultaneously. For picture refreshing, a display file must be continuously available to drive the console CRT.[1] With multiple terminals, there will not be room in the satellite's core for all of these necessary files either. These considerations lead to the conclusion that the display data structures and at least a sub-library reside on a local mass storage device so that particular items needed at any time can be obtained relatively quickly. In addition, the display files for picture generation and refreshing must reside on some mass storage device, preferably, one that can automatically refresh the pictures on all consoles without use of the satellite CPU. This last notion, in turn, necessitates a controller to interface the satellite CPU with a mass storage device and the individual terminals, performing such functions as regenerating the displays, changing appropriate display files when requested, and queuing all terminal interrupts. If possible, one mass storage device for all the functions mentioned (library storage, data structure storage, display file storage) would be preferred. The resulting hardware configuration desired is presented in Figure 1.

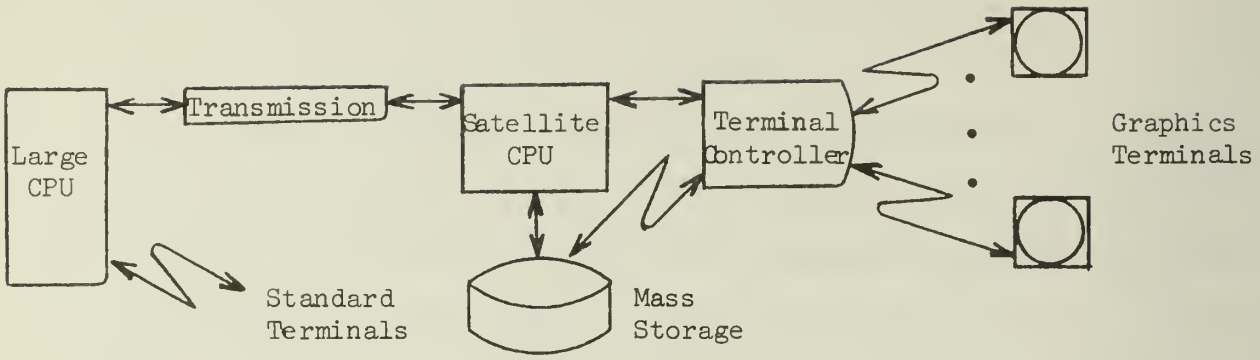

Figure 1.

Hardware Configuration

Since graphics consoles are being used, the acceptable class of problems should be anything that can be represented on the display surface for which the user can describe the various elements and their interconnections. Thus, for example, the system should handle an electrical network, or a piping network, or a bridge, or two levers, or a horse race, or a glop of cytoplasm, assuming that the respective users can supply the appropriate equations, parameters, and constants. A somewhat detailed description of one way of providing this information is presented in a paper by C. W. Gear cited in the Bibliography. In general, the system should allow a user to create any kind of a pictorial element, define its action with any combination of equations, constraints, connection points, constants, parameters, etc., and then use these elements to represent his particular problem. Of course, if a set of elements such as resistors, capacitors, etc., for a particular type of problem has been previously defined, this set will be available in the library system for other users, who simply connect them as appropriate and supply new parameters. The data structure representing the completed problem statement is then sent to the central CPU for analysis.

In order to provide analytic flexibility in the central CPU for such a problem class, two approaches can be taken. First, many analysis programs for certain restricted types of problems currently exist. These programs usually assume some type of symbolic input and range from non-interactive to highly interactive. A table-driven or compiler-compiler designed symbolic manipulation interface program could scan the problem data structure, and create symbolic input for whichever analysis package the user may have specified. This would make a fast special purpose package, rather than a slower general package, available for the problem--assuming such a package for the problem exists at all. The output from these programs, whether interactive or not, could then be sent to an applications drawing program which would produce a data structure suitable for creating a display on the terminal. So the input, analyze, and output cycle is complete.

On the other hand, packages do not exist for many problem types. In this case, a general purpose package (differential equation solver) must be used.[2] But here the symbolic manipulation interface need only produce one type of output, although the type of syntax and other checking performed by it may be considerably more complex than in the case described above. The output from this type of analysis package would also be passed to an application drawing program to communicate results to the terminals. Actually, there is no reason why both approaches cannot be used in the system, leaving the choice to the individual user as best suits his needs.

The software environment in which the three essential analysis components (namely, symbolic manipulation interface, analysis packages, and applications drawing package) must operate has been stated and implied in this and the previous section. To review and restate briefly, 1) communications and control executives must be in operation in both the central and satellite CPU's, 2) an information retrieval facility must be available to both CPU systems with a permanent main library in the central computer and an optimized temporary sub-library in the satellite, and 3) a data structure management and display file generation facility with drawing, picture/subpicture, text, and prompting facilities must be available in the satellite for direct communication with each terminal user. In addition, the executive system of the central CPU must utilize temporarily unused resources allocated to support of the satellite in order to improve the efficiency of the central CPU. One way of accomplishing this last requirement is to provide for the running of background utility packages whenever space is not being entirely taken up by symbolic manipulation, analysis, or drawing packages. Explicitly, terminal users could send commands to a sub-monitor to operate on some previously constructed symbolic file. This file contains control and source information similar to a batch input stream, and the sub-monitor schedules translators and processors for execution accordingly. If communication were established between the graphics support executive and the time-sharing executive, this facility would be opened to all non-graphics terminals in the system, and the files used could be conveniently located in the time-sharing filing system. Furthermore, certain types of problems can be stated on an ordinary keyboard type time-sharing terminal, such as algebraic analysis, language translation, etc. This input could

be interfaced to other analysis programs in the simulation system. Hence, the system could support the maximum facilities that any particular terminal can handle, while increasing the availability of the system from a limited number of graphics users to all terminal users. The desired software configuration is presented in Figure 2.

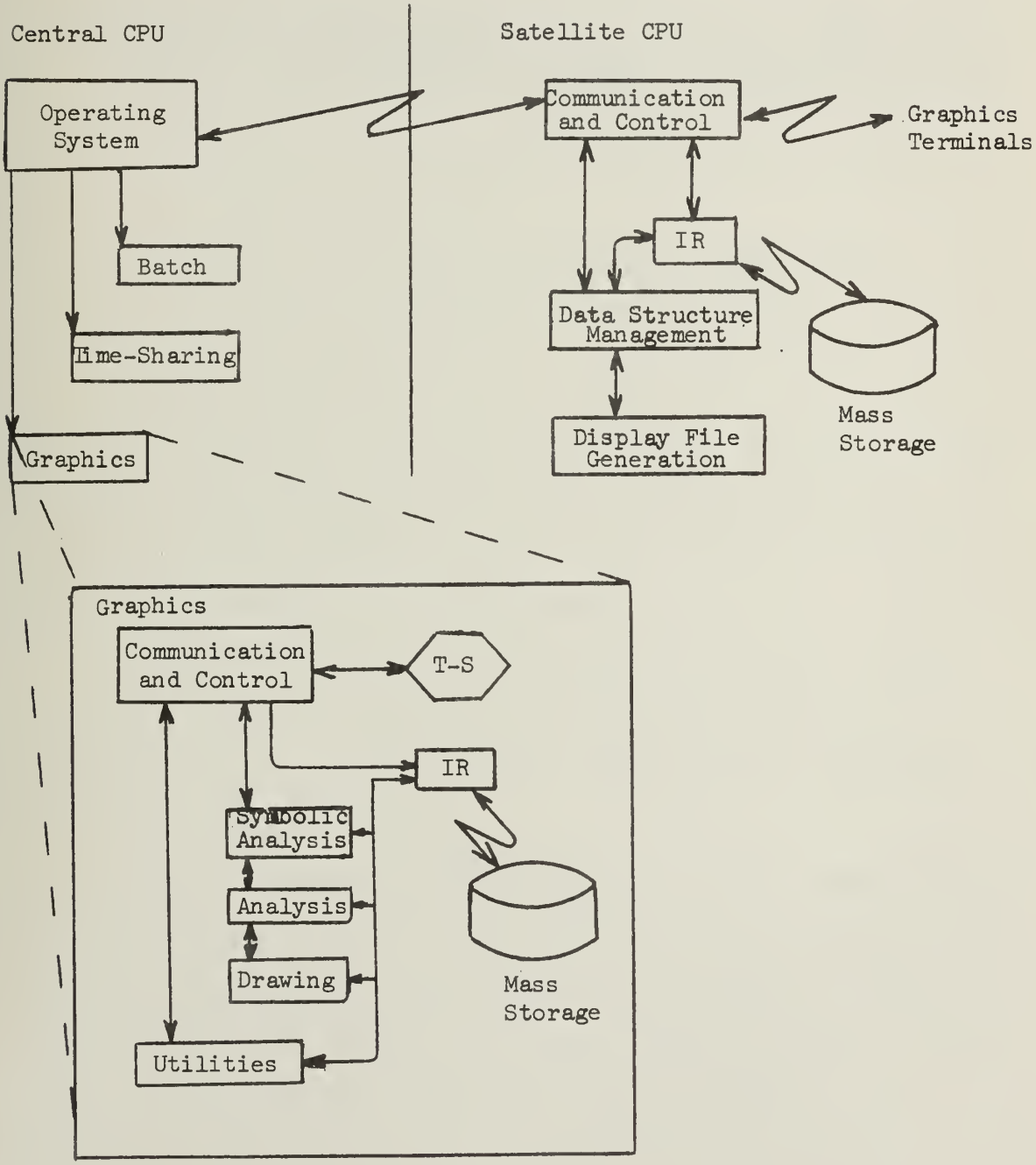

Figure 2.

Software Configuration

FOOTNOTES

[1] An alternative is to use storage tubes, but these add the problems of resolution, decay, inability to partially update a picture, erase time, higher cost (typically), etc. However, some systems using this technique have been built, particularly the one by Engelbart, Stanford University.

[2] A good example of a state-of-the-art general differential equation package is the ODESSY system under development at the University of Illinois, Department of Computer Science (c.f. Dill, C. et al.).

III. PROPOSED SYSTEM--AN OVERVIEW

A. Hardware

The GRAS System, currently under development to meet the previously mentioned requirements, utilizes the following hardware in a configuration identical to that in Figure 1:

PDP8 as the satellite CPU

Display Processing Unit (DPU) as the terminal controller

Eight Graphics Terminals with joystick, keyboard, and function keys

Head per track disk as the local mass storage device

2701 Parallel Data Adapter for data transmission

360/75 as the central CPU

In particular, the PDP8, 2701, and 360/75 are only being used in the prototype system because they are available; the disk, DPU, and terminals are inhouse pieces of equipment whose designs have been geared to the requirements of the project.

The PDP8 is a general purpose computer with four 4k banks of 12-bit per word core memory and I/O interrupt facilities. It interfaces with the DPU, disk, and 2701 as well as with a teletype console and a low speed printer.

The DPU takes a standard display file (in this case, 338 code) from PDP8 core and executes it, thus producing a 3-bit incremental code to drive a terminal CRT. Decoding logic at the terminal interprets this 3-bit code, moves the CRT beam accordingly, and thus produces a visible image. The display file is "executed" by the DPU only once; the resulting incremental code is stored on a track of the disk (each terminal

is associated with one particular disk track). Subsequent refreshing is done directly from the disk by feeding the contents of the appropriate track to each terminal's decoding hardware, all simultaneously and without interrupting the CPU or using CPU core. The DPU also queues interrupts from the various terminals and passes this information to the PDP8 on demand. Hence, different displays can be continuously serviced, changed, and refreshed at a multiple number of terminals without overloading the PDP8's interrupt structure or available core.

Each terminal has a CRT, joystick, keyboard, and function keys. The joystick, which replaces the use of a light pen for inputting XY coordinate information, has a locally produced (i.e. local to the terminal) cursor indicating its position on the screen. However, an interrupt occurs, making XY information available, only when a button is pressed. So the user can move his cursor all over the screen without bothering the CPU until he really wants to. Unlike a light pen, the joystick can be used to input XY information from blank areas of the screen; hence, the pen tracking problem, and the associated high density of interrupts is eliminated. A "dummy generation" (i.e. comparator) mode is available in the DPU, enabling access to display file status and addresses in the manner usually available with standard display devices. (The joystick could be directly replaced by a data tablet using a discrete interrupt type pen.) Keyboard input is buffered and displayed locally, one line at a time. That is, the user can view the current line being typed and will interrupt the CPU only when the line is completed. As previously mentioned, all terminal interrupts are filtered through the DPU before reaching the PDP8.[1]

The disk is a high capacity, high rate device with head-per-track hardware. Each of its 32 tracks contains about 100,000 bits--the equivalent of 16k PDP8 words (i.e. all four banks). The transfer rate is 2 usec. per word, so an entire bank (4096 words) can be brought into the PDP8 in an average time of about 25 mlsec. (at 1800 rpm yielding an average rotational delay of 17 mlsec.). Due to the head-per-track feature, all displays can be simultaneously refreshed, as mentioned in the DPU section. With hardware and software sectioning, this one disk performs all three mass storage functions mentioned in Section II. The assumption of eight graphics terminals is made for the following discussion, and it should be noted that each disk track is hardware divided into 12 sections.

Eight of the 32 disk tracks are hardware allocated to the DPU for the refreshing of the eight graphics terminals. Each of the tracks appears to the respective terminal as a continuous stream of 100,000 bits (33,000 incremental codes). Eight other tracks are software allocated for full-bank task switching. This space is used to save the current picture data structure, display file, and program status for each terminal as swapping in and out of core becomes necessary to service the various terminals. One track is software allocated for loadable systems programs needed in the PDP8. Each program is located at a specific track position for rapid read-in. The remaining 15 tracks are software allocated for library space. On these tracks, each sector is viewed as containing ten blocks of 128 PDP8 words each, or 1800 total blocks for the 15 tracks. Since reading and writing on the disk is possible by sector only, an additional "selective read" hardware facility has been

added. This allows any pattern, and particularly, as few as any one, of the ten blocks in a sector to be read sequentially into the PDP8. For example, blocks 0, 1, 5, and 8 in sector 4 of track 10 could be specified, and blocks 0, 1, 5, and 8 would appear contiguously in PDP8 core. Hence, since reading is the dominant operation involved in creating a display file, selective access to particularly small blocks in a relatively large local data base offers a distinct saving in computer time.

The 2701 is a standard high speed transmission device with a data rate of about 0.5M cps. At this rate, an entire PDP8 bank can be read-in or written-out in about 50 mlsec, making significant data exchanges between the two CPU's relatively comfortable.

The 360/75 (under MVT), while supporting standard batch and time-sharing facilities, provides a permanent region for supporting the satellite CPU. In this region, computational power as well as access to extensive 2314 disk storage is available. Effective utilization of these resources while not in use for PDP8 requests is discussed later in this section.

B. Software

The GRASS software configuration is similar to that in Figure 2. A description of each component is presented after the following summary of a typical user's session.

Support for the satellite in the central CPU and the satellite operating system have been previously initialized, and several users at other graphics terminals are variously creating elements, stating

problems, requesting analysis, and viewing results. The new user activates his terminal and "signs-on" with valid identification information. After requesting that certain subsets of his personal library and of the general library be transferred to the local system, he constructs some new elements (personal), changes some old elements (personal), and then begins defining a problem situation with both old and new, personal and general, elements. Later, he directs the system to update the central library with the new and altered items, to save the problem situation (really just an element in the library), and to purge the local system of the library items he requested (courteously leaving more local space for users still defining problems). Since he asks and finds that space exists in the background-utility job queue, the user passes the system the name of a batch-type file to be executed (he created this file at a standard time-sharing keyboard console that morning). The user then specifies the interface and analysis packages that are required for the problem he has just defined and requests analysis of that problem to begin. While awaiting the results, he is notified that his background job bombed due to incorrect control cards at statement 137. Requesting service as a temporary time-sharing console, the user corrects his file; he exits from time-sharing mode and finding the background job queue again low, re-enters his job. The results of the requested analysis have become ready, so he examines them, decides to change some parameters, and analyzes them again. This process continues until a satisfactory result is obtained, after which the user requests hardcopy output (which is produced at some later time, offline). The user requests that his new problem be added as an element in the general library and "signs-off".

When an interrupt occurs at a terminal, indicating that servicing of the terminal is required, that interrupt is processed on several levels. First, the DPU tests to see if the interrupt type has been enabled, and if it was not, the interrupt is ignored (joystick inhibit, pushbutton disable, keyboard disable, etc.). Otherwise, the interrupt is queued by the DPU for processing by PDP8 software whenever the PDP8 is free. At the next level, when the PDP8 begins processing the interrupt, the PDP8 executive program first tests a software status mask for the terminal. This mask, set by some program operating under the executive and currently handling communications with the terminal, can specify whether the interrupt is a Class I or Class II interrupt. Class I interrupts merely reset certain bits in the mask or raise various software indicators for the terminal. Processing of these interrupts (such as keyboard characters, or perhaps certain pushbuttons, etc.) only requires the executive, which clears the interrupt condition when finished. Class II interrupts (such as joystick, end-of-line character, some other pushbuttons, etc.) require task switching. In this case, the required program (if not already present) and associated data structures for the terminal must be brought into PDP8 core from the program (1 track) and swapping (8 tracks) areas of the disk. At the final interrupt level, when the swap is completed, control is passed to the lower level communications program which examines the interrupt information in detail and performs the necessary operations (usually changing the picture at the terminal). This program then exits to the executive and processing of the next Class II interrupt begins. Note that while the Class II interrupt was being processed, the executive could still process all Class I interrupts that might have occurred. When the

lower program returned to the executive, it specified which banks of core were to be saved for the terminal and any changes desired in the terminal mask.

The above chain of events for interrupt processing in the satellite implies the activities of the major software components of GRASS at that end of the system. The executive (GLASP) is responsible for all interrupt scheduling in the PDP8. GLASP maintains the terminal masks and other terminal status information (current program, banks saved, etc.), initiates task switching, and provides software linkages between lower level programs and system utilities (remote data transmission, library requests, print requests, etc.).

The lower level program (FLOG) manipulates picture data structures, creates display files, initiates action in the central CPU, and prompts the user at the terminal. Whether the data structure was created by user interaction with FLOG on the PDP8 or by a program in the central CPU and then transmitted to the PDP8 is irrelevant. Hence, FLOG is the two-way link between the terminal user and remote analysis or utility packages in the central CPU.

When FLOG needs data from the library--usually during generation of a display file--a call is created to the satellite portion of the information retrieval facility (GIRLS). Since the storage capacity of the disk at the PDP8 is relatively limited, the complete permanent system library must be located at the central CPU. Normally, requests to this permanent library are made by interface or application programs in the central CPU or by the portion of GIRLS resident in the PDP8. Use-count, protection, and hierarchy information in the central library is combined with space and contents information about the local disk whenever the PDP8

makes a library request. As mentioned, a user typically will request his own sub-library plus other subsets of the system library at the beginning of a session. The most often used items--consistent with protection, space availability, and items already on the local disk--will then be transferred to the PDP8. Thereafter, most items needed by the user will be readily available locally, and only occasional references to the permanent library will be made. Hence, GIRLS attempts to optimize retrieval in a double ended, unbalanced system.

Swapping and program requests from GLASP and specific library requests for GIRLS are handled by a disk access software package (DOOFIS). DOOFIS provides routines for controlling the program, swapping, and library sections of the disk by sequential-block or track-sector-block numbering schemes. Inline coding is used to move entire swap banks or program code at high speed on single disk transfers into the PDP8. Other routines handle selective-read (single transfer) or blocked-read (buffered transfer) and blocked-write (buffered transfer) requests to the library section of the disk.

The flow of control in the support region of the central CPU is similar to that in the satellite. All input from and output to the PDP8 passes through an executive (GASP). Each input line from the PDP8 has a terminal identification character in it, and each graphics (or other) terminal has a control block associated with it. When an input line is received, GASP checks the respective control block (as indicated by the ID character) to see if a user has successfully "signed-on" yet. If one has not, the line is passed to a "sign-on" monitor for inspection; if one has, the block is checked for the presence of an attached subtask. If a subtask <u>is not</u> present, the line is sent to a control monitor. This

monitor performs functions for the terminal, such as passing commands to the background job queue, creating subtasks (any interface, analysis, applications, etc., programs), passing requests to the central CPU portion of GIRLS, and passing lines to the time-sharing executive. If a subtask is present, the line is enqueued for later passage to it (a special control character, however, can be used to cause the input line to be sent to the control monitor even when in "subtask mode"). Depending on the amount of central CPU core available, any number of systems terminals can be in any combination of control or subtask mode, with the background job processor running as well. Hence, if all terminals were "signed-on", but in control mode, the background job processor would be making use of practically all of the resources allocated to graphics support--resources that ordinarily would be idle.

The central CPU section of GIRLS is a permanent subtask of GASP, thus making it continually available for PDP8 requests or central CPU program requests. It contains all routines for updating, extracting, and monitoring (use-count protection, etc.) information in the permanent library.

Although any program can be attached by GASP as a subtask for a terminal, the usual package contains an interface, analysis, and applications group; or a sequence of attaches will bring such a grouping into execution consecutively. This allows complete flexibility for the user to determine what interface and analysis routines he wishes to apply to his particular problem.

An interface (SYMP) takes a data structure as input, converts it to symbolic strings or another data structure as necessary (depending on the type of analysis program that will be used), and checks the result for

syntactic correctness. In converting the input data structure, SYMP may make calls to GIRLS to evaluate items used as sub elements. The finished information is passed directly for analysis or stored in a data set for the next attach (i.e. the analysis program is not yet in core).

The analysis program (GEAR) can be of any type, accepting data in structured or symbolic form, interactive or non-interactive, etc. As output is produced it is sent to an applications program to be reconverted to a data structure suitable for input to FLOG in the PDP8.

Applications programs (GADS) used in the system can have picture-subpicture facilities and access to GIRLS, thus making them more powerful than most packages of this type. Optimally, there would be only one GAD used by all GEARs.

To summarize briefly--by using generalized FLOG, SYMP, and GADS packages to link users with GEAR or the utility package, GRASS attempts to enlarge the applicable problem class and eliminate much of the inflexibility prevalent in most current systems. By utilizing some special hardware (disk, terminal controller, and graphics terminals) in conjunction with a satellite-central CPU configuration operating under suitably designed executives, GRASS attempts to make a powerful graphics system available at a reasonable cost per terminal, while maintaining a high degree of dedicated service at the terminal and a high efficiency rate at the central CPU.

FOOTNOTE

[1] For a complete description of the functioning of the DPU and for additional information about the disk, consult the Masters Thesis by R. Hostovsky, Department of Computer Science, University of Illinois, Urbana, Illinois.

IV. GASP/GLASP

A. Current Implementation

The desired prototype configuration will not be available until sometime after January 1970. The DPU and terminals are in the design stage, and the disk hardware is being debugged and the software for it implemented. Access to a large CPU is restricted to a 360/50 via a low speed link through a PDP7 until the arrival of another 2701 PDA.

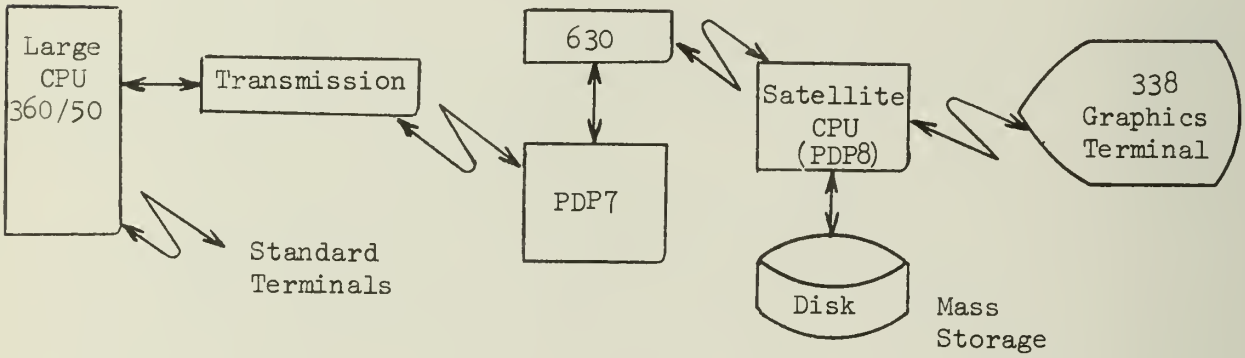

Figure 3.

Current Hardware Configuration

Interim versions of GASP and GLASP (hereafter referred to as GRAFMON and UIMON8, respectively) using the available configuration of Figure 3 have been implemented. Items to note are:

1. Only a single graphics terminal is attached to the PDP8.
2. Only a restrictive, low speed data link to the large CPU is available.
3. The large CPU is operating under MFT with a resident ASP system.

This interim communications system enables graphics programs in the PDP8 to interact with support packages in the 360/50, although the effectiveness and flexibility allowed is not comparable to that which will be typical in the completed system.

After initiating GRAFMON support in the /50 CPU, the user brings UIMON8 into PDP8 core. He can then communicate directly with GRAFMON over the PDP8 teletype console, directing it to bring a specific module into the /50, to pass a data set to time-sharing, or to load a data set into PDP8 core to run under UIMON8. Typically, the user would load a drawing program into the PDP8 and a library module into the /50. He would then proceed to construct a problem situation on the 338; the completed data structure would be transmitted to and stored by the library module. Control would be returned to UIMON8 and GRAFMON, respectively. The user then requests an interface, analysis, and applications module (or requests, in order, each one separately) to perform the desired analysis whose results are stored in the /50 library. The process then continues in this fashion until satisfactory results are obtained.

B. 360 Section

1. General Description

GRAFMON support for the 360/50 graphics area consists of a non-terminating monitor and a comprehensive set of communications, translation, and data definition macros for programs running under the monitor.

The monitor consists of three segments: initiator modules, a permanently resident SVC, and a communications module. The initiator module uses the SVC to save information about the graphics partition on a disk data set. Then the initiator overlays itself with a very small bootstrap module, which in turn brings in the communications module. Data transmission between the 360/50 and the PDP8 is now possible, and the PDP8 can request certain functions, such as data set transmission, linkage to the Time-Sharing partition, and load module execution. To execute a load module (e.g., analysis, interactive, or utility programs), the communications module overlays itself with the desired module. Upon normal termination of the called module, control is returned to the bootstrap, which loads in a fresh copy of the communications module, and the process continues. If, however, the called module abnormally terminates, control is passed to the MFT ABEND module. Due to the SVC, and some changes in ABEND, the graphics partition is rebuilt using the information stored by the initiator on the disk. Control is passed back to the bootstrap, and processing can still continue as if the ABEND had not occurred.

2. Macros

The following macros are available to 360/50 programs for communication with the PDP8 via the PDP7:

PDP7IN	--call PDP7EXCP to input data
PDP7OUT	--call PDP7EXCP to output data
PDP7TR	--call PDP7EXCP subroutine for data translation
PDP7EXCP	--perform I/O operations
nDC	--assembly time macro for defining data and character strings

A complete description of each macro is contained in Appendix A; a brief description follows here. A detailed summary of 360-PDP7-PDP8 communication characteristics is contained in Appendix C.

PDP7IN creates a calling sequence to an I/O routine in PDP7EXCP. This sequence specifies the data input location, the amount of data expected, and the addresses of error routines. Similarly, PDP7OUT creates a corresponding sequence for data output. As mentioned, PDP7EXCP contains the actual I/O routines needed for communication. These routines start the I/O operation, wait for completion, check for errors, and move data as appropriate between user areas and their own buffers.

PDP7TR provides the data translation facilities necessary for converting 360 formatted data into PDP7-PDP8 formatted data (c.f. Appendices C and D). The user specifies one of eight translation types, the location of the data, and the length of the data. Translation is performed in place.

nDC defines character and numeric data at assembly time in PDP7-PDP8 format. Hence, data known to be needed in advance does not have to be translated during execution. This is particularly useful for specifying error messages, display files, etc. The macro has default input and output character sets, but these may be overridden by the user.

C. PDP8 Section

1. General Description

The UIMON8 monitor system provides routines for communication among PDP8 programs, 360 programs, and users at the PDP8 teletype console, as well as facilities for the general convenience of all users.

UIMON8 resides entirely in bank 3 of the PDP8. Also, all of bank 1 is used as a buffer area when the text editor is being used, and the first three words of bank 0 are used to link to the interrupt handler. Locations 0 through 4777 of bank 3 contain the UIMON8 program. The area from 5000 through 5777 is reserved for user routines (page zero of bank 3 is completely filled with constants and save areas for the UIMON8 system; users may use the constants and pointers, but may not use any of the storage areas) and the area from 6000 through 6777 contains the character generator (part of the remaining area is "FREE" space to be used by a larger character generator with the last couple of pages in core being reserved for the resident routines of the future disk facilities). Programs running under UIMON8 operate in banks 0, 1, or 2; use the UIMON8 interrupt handler for interrupt processing; and uses the UIMON8 transmission routines for sending and receiving data.

After UIMON8 is in PDP8 core, it can be directed to enter one of several modes. The first allows direct teletype communication with GRAFMON. In this mode, the user specifies modules to be loaded into the /50, into PDP8 core, or sent to time-sharing. Another provides a local graphics text-editing facility for building symbolic files. These files can be printed on the teletype, stored on magnetic tape locally, or sent to the time-sharing filing system. A third mode stores specified PDP8 core images in time-sharing files. These "programs" can then be reloaded into the PDP8 at a later time and executed. Finally, the 338 can be used as a high speed graphics time-sharing console.

When UIMON8 "exits" back to the PDP8 operating system, only control has been lost; the program itself is still in bank 3. Hence, the user can specify another PDP8 program to be loaded into banks 0, 1, or 2. This new program, in turn, simply restores words 1, 2, and 3 of bank 0 (for the interrupt handler) and turns the PDP8 interrupt hardware on. The user program is now running "under" UIMON8 and may use any of its communication and interrupt facilities.

2. Routines

The following routines are available to programs running under UIMON8 in the PDP8:

For processing all individual device interrupts

INTHND

For sending data to the 360 system

ASYSMSG

PSYSMSG

For receiving data from the 360 system

MSGES

LOAD

For communicating with the user via the teletype

SYSMSG

SYSERR

The system interrupt handler is table driven, utilizing two tables: one with clear-flag or test-flag IOT's and the other with corresponding device routine addresses. Hence, to ignore interrupts from a particular device, it is merely necessary to change a test-flag IOT in the first table to a clear-flag IOT. Then, if an interrupt occurs from that device, the flag will automatically be cleared and a branch to the device routine will not be initiated. The converse procedure is used to accept interrupts from various devices. Clearly, if a user desired to handle interrupts from a particular device (such as the light pen) himself, all that is necessary is for him to save the system routine address in the second table and substitute the address of his own routine. This is one of the main reasons for reserving part of bank 3 for user routines, i.e. for use by user's interrupt routines. Hence, each user who wishes to use any of the peripheral devices of the PDP8 need not construct his very own interrupt handler, but need only write routines to handle interrupts from devices for which system routines do not meet his special needs (c.f. description of interrupt tables, Appendix E).

For communicating with the 360 system two buffers for input and output, named respectively SBUFF and UBUFF, are maintained along with several flag words. A special "FILTER" routine (GMON8) looks at every input line from the 360 to see if it begins with a valid control character. These control characters cause the input line to be passed to a particular routine associated with each character. Hence, a form of message switching is obtained; that is, information can be sent intermittently to various routines in the PDP8. As an example, assume an interactive graphics program is running in the PDP8 with an analysis or interpreter program in the 360. If the 360 sent a line to the PDP8 with an invalid control character, the line (assumed to be a system message) would be automatically printed on the teletype and the user could type a coded response (c.f. description of SYSERR routine, Appendix B). The 360 program could have specified the control to call "LOAD", in which case the input line is assumed to be data in core load format. This type of input (being data directed in nature) could have replaced part or all of the current display file, set switches for the executing program, or could have consisted merely of input data to go into a user's own input buffer, or do anything else that a clever user might desire. Or, the 360 might have sent the control for entering the debugging package; or the 360 might have sent the control indicating that the input line was to go to the current input routine which might have been some other system routine or a particular routine

of the user. All of the above input operations are performed with no effect on the currently executing program in the PDP8. In addition, since the control characters are associated with routines via a table, the user can add new ones or redefine old destinations, etc.

It is hopefully obvious at this point that input from the 360 to the PDP8 is handled practically on an automatic basis with the main constraint being that the input data is of the proper format. Hence, the major programming of input formatting winds up being done in the 360, where clearly programming is much easier for the average user. (In addition, when FORTRAN and PL/I interfaces are provided at the 360 end--work in progress--easy communication will exist for just about anyone.)

For transmitting data to the 360, things are a bit more complex, but equally as flexible for the PDP8 user. To send core image data, i.e. data that may represent any bit pattern, the routine PSYSMSG can be used. This routine can be called from any bank and will send a core block to the 360 as specified by the length, bank, and starting address information supplied in the calling parameters. Each PDP8 word is converted to two 8-bit characters to insure that no PDP7 control characters are accidentally transmitted (c.f. description of PDP8 data formats, Appendix D, and PSYSMSG routine, Appendix B). To send ASCII character data to the 360, the routine ASYSMSG is provided. This routine assumes that UBUFF contains a control character of the user's choice followed by the ASCII message (one

character per word), followed by a carriage-return. The utility of this routine is made much clearer by the descriptions of the SYSMSG and MOVEDT routines.

V. CONCLUSIONS

Although the system outlined in this paper and the sections implemented so far indicate to the author that this will prove a viable solution to the problems discussed, it is, unfortunately, still too early in the implementation stage to state conclusively that the final system will indeed prove successful.

BIBLIOGRAPHY

Blake, K. and Gordon, G. "Systems Simulation with Digital Computers," IBM Systems Journal, Vol. 3, No. 1, 1964.

Dill, C., Ellis, C., Gear, C., and Ratliff, K. "The Automatic Integration Package for Ordinary Differential Equations," Department of Computer Science File No. 779, University of Illinois, Urbana, Illinois, 1968.

Gear, C. W. "An Interactive Graphic Modeling System," Department of Computer Science File No. 318, University of Illinois, Urbana, Illinois, 1969.

Harmon, H. H. "Simulation as a Tool for Research," Systems Development Corporation, Santa Monica, California, SP-565, 1961.

Hobbs, L. C., ed. "Progress in the Computer Field," Computer Group News of the IEEE, Vol. 1, No. 7, July 1967.

Hostovsky, R. "Design of a Display Processing Unit in a Multi-Terminal Environment," Masters Thesis, Department of Computer Science, University of Illinois, Urbana, Illinois, 1969.

Shigley, J. E. "Simulation of Mechanical Systems," McGraw-Hill, New York, New York, 1967.

APPENDIX A

360 Macros

For all macros, ØS/360 linkage conventions apply. A detailed summary of 360-PDP7-PDP8 I/O conventions, formats and restrictions appear in Appendix C.

MACRO CALL:

label PDP7IN AREA=label,LEN=m,MSG=label,ERR=label,SOP=label,EXT=Y

Description:

This macro causes the next input record from the PDP7 to be put into core at the location specified by the AREA operand. The amount of data passed to the user is equal to the (HEX) number of bytes specified by the LEN parameter plus one, the last character being a carriage-return (X'8D"). If 80 column card images were expected, LEN=80 should be coded and 81 characters would be transferred. (When scanning an input string, the user need only scan for the carriage-return rather than keep a character count. Also if the string were shorter than expected, a carriage-return from the PDP8 will appear earlier in the record indicating the end of the line. Of course, if the record length is unknown, a request for a maximum length record is best.) LEN must be $\leq$ 125. Both AREA and LEN may be specified as residing in registers, e.g. AREA=(5) means the address of the input area is in register 5. LEN=(6) means the length of the input request is in register 6.

The input operation can have three possible results: success with data received, success with a system code received, or EXCP channel-error (very rare). Three corresponding parameters may be coded. For any not coded control passes to the next sequential statement after the calling sequence.

SOP=lab --control passes to "lab" if data successfully received.

MSG=lab --control passes to "lab" if a system message is received. In this case, bytes 0 and 1 (A and B) instead of data bytes are placed in the first two bytes of the user's input area for examination.

ERR=lab --control passes to "lab" if channel-error occurs. Best action is to assume line lost and indicate same to user with sysmessage facility. Hence, effect retry.

If EXT=Y is coded, the appropriate linkage for an externally defined PDP7EXCP CSECT will be generated (c.f. PDP7EXCP macro).

MACRO CALL:

label PDP7OUT AREA=label,LEN=m,ERR=label,SOP=label,EXT=Y,

SYS=nn, MON=nn

Description:

This macro causes the output record at the core location specified by the AREA operand to be sent to the PDP7. The amount of data passed to the PDP7 is equal to the (HEX) number of bytes specified by the LEN parameter. LEN must be $\leq$ 124. Both AREA and LEN may be specified as residing in registers (e.g. PDP7IN).

The output operation can have two possible results analogous to the read operation: success with data sent, or EXCP channel-error (very rare). (c.f. PDP7IN for a description of ERR and SOP.) The action of EXT=Y is identical to the PDP7IN case.

The data line sent to the PDP7 is formatted:

```
AB[NX...X CR--Junk--]CR
01 23   k k+1 k+2 126 127
```

where A and B are PDP7-360/50 system communication bytes, N is the PDP8 graphics monitor byte, bytes 3 through k are the data bytes, byte k+1 is a carriage-return (X'8D') inserted by PDP7OUT, bytes k+2 through 126 are junk from previous operations (these are ignored by the PDP7), and byte 127 is a mandatory carriage-return (X'8D') inserted by PDP7OUT. If 124 data bytes had been specified, there would be no "junk" bytes, and the carriage-return at bytes k+1 and 127 would coincide at byte 127.

The SYS and MON operands are used to specify bytes B and N.

For SYS:

00 = Time Sharing OFF

01 = Time Sharing ON

02 = Log In

03 = Log Out

04 = Data Line

For MON:

84 = Input Data

A3 = System Message

80-83, 85-86 = System Options

NOTE: User default options are SYS = 04, MON = 84.

The user may specify MON = A3 for transmitting lines to the system message device. All other codes are for systems maintenance and are password protected at this time.

MACRO CALL:

```
PDP7EXCP   EXT=Y,DDNAME=PDP7DD
```

Description:

This macro actually performs input/output operations with the PDP7 when called via PDP7IN and PDP7OUT. PDP7EXCP must be present if either PDP7IN or PDP7OUT is used, unless EXT=Y is coded for PDP7IN and PDP7OUT. Then PDP7EXCP with EXT=Y must appear in another CSECT in the same load module. The DDNAME operand specifies the name of the DD card that indicates UNIT=PDP7 to the operating system, with the default DDNAME being PDP7DD.

PDP7EXCP should be the last statement in an assembly or if it is not, any statements that come after must be named as a new CSECT.

MACRO CALL:

label PDP7TR n,LAB=label,LEN=m,EXT={X,Y}

Description:

Depending on the first operand, n, this macro translates blocks of data ($\leq$ 256 characters) at run time from one format into another. The LAB operand specifies the location of the block to be translated, and LEN specifies its length in (HEX) bytes. Both LAB and LEN may be specified as registers (c.f. PDP7IN). On the first occurrence of a call for each type of translation where the EXT operand is not coded, the translation table and/or routine is generated in addition to the calling sequence. Each subsequent call for a particular translation type results in only the calling sequence being generated. If EXT=Y is coded, only the calling sequence is generated and the translation table and/or routine is assumed to be externally defined. If EXT=X is coded, only the translation table and/or routine is generated and calls are assumed to be externally defined. (NOTE: PDP7EXCP and all the translation tables and/or routines could be put in one central "input/output" CSECT. Other CSECTS would only need to use the calling sequences, PDP7IN, PDP7OUT, and PDP7TR.

There are eight translation types defined at present:

n = 1 --translates from EBCDIC to ASCII.

2 --translates from EBCDIC to coded (6 bit) ASCII.

3 --translates from EBCDIC to coded (6 bit) BCD

4*	--translates from /360 half words (right 12 bits) to pairs of coded (6 bit) binary.
5	--ASCII to EBCDIC.
6	--coded ASCII to EBCDIC.
7	--coded BCD to EBCDIC.
8*	--coded binary pairs to /360 half words.

For n = 1, 2, 3, 5, 6, or 7 carriage-return (X'8D') and line feed (X'8A') are preserved by the translation. For example, if X'8A' is the last byte in an EBCDIC string being converted to 6 bit coded ASCII (n = 2), the corresponding byte in the resultant string is still X'8A'.

* LEN is the number of half words to be transmitted.

MACRO CALL:

label nDC 'message',{OWN=}

Description:

This macro allows data definition at assembly time of ASCII, coded ASCII, coded BCD and octal data, depending on the character n.

Forms:

*n = A	--define the EBCDIC characters enclosed in apostrophes as ASCII character data (i.e. lab ADC 'IJK' becomes lab DC X'C9CACB').
*n = B	--define the EBCDIC character enclosed in apostrophes as coded ASCII character data (i.e. lab BDC 'ABC' becomes lab DC X'828486').
*n = C	--define the EBCDIC characters enclosed in apostrophes as coded BCD character data (i.e. lab CDC 'ABC' becomes lab DC X'E2E4E6').
n = Ø	--define the octal numerals as four digit octal numbers in 6 bit coded binary (i.e. lab ØDC 173 becomes lab DC X'82F6').
n = DØ	--define the decimal number as one four digit octal number in 6 bit coded binary (i.e. lab DODC 123 becomes lab DC X'82F6').

* " or && used to define ' or & may <u>not</u> be split into two card images.

The OWN operand is valid with n = A, B, C, only and can be used to redefine the output character set of an nDC definition. The OWN operand specifies: 1) that an internal redefinition is being performed (and not a data definition), and 2) the number of characters in the message that replace each character in the output character set. For example, ADC 'C1010203',OWN=2 would cause the output equivalent of B, C, and D for subsequent ADC definitions to be changed from C2, C3, and C4 to 01, 02, and 03, while keeping C1 for A and all the other character definitions as previously defined as well, i.e. before ADC 'ABCD' became DC X'C1C2C3C5'; now, ADC 'ABCD' becomes DC X'C1010203').

Also, the user may change the standard character set if he desires by use of the STANTAB L, message macro. For example, STANTAB 1, '345' would change A, B, and C to 3, 4, and 5, i.e. formerly ADC 'ABCD' became DC X'C1C2C3C4' now ADC '345D' becomes DC X'C1C2C3C4'). Hence, the user may completely alter the character definition facilities if so desired. (NOTE: 1) any character set may not exceed 150 items, 2) replacement by use of OWN is character by character beginning with the first character group in the OWN "message" replacing the first character result in the output set and continuing sequentially only until the "message" is exhausted. Hence, resultant groups in the output set not affected by the OWN "message" remain as previously defined. Users must exercise care in changing the character sets to insure that mixed length definitions within a set are not created, 3) error checking is difficult and expensive for this macro and is not extensive: users are urged to be very careful and follow procedures for its use closely).

APPENDIX B

UIMON8 Routines

All user-oriented routines in the UIMON8 system assume a JMS type call with the "IF" field set to 30 and the "DF" field set to the calling bank, i.e. the called routine will return to the user's program with the "IF" and "DF" fields set to the value of the "DF" field at the time of the call.

ROUTINE: ASYSMSG

FUNCTION: Send a line of ASCII characters to the 360

CALLING SEQUENCE: ACC = Number of characters to be sent
(including a trailing carriage-return)

DESCRIPTION:

The routine assumes that a line of ASCII characters with one character per word is already in UBUFF. This line is sent to the 360. If an error in transmission occurs, or if at sometime during the transmission the 360 sends a message to the PDP8, this message will be printed on the teletype followed by the sequence "CODE:". If the user types a zero, the transmission operation will be terminated. However, any other character will cause the operation to be retried.

UBUFF may be filled by the user with the aid of the "MOVEDT" routine or the SYSMSG facility (c.f. the descriptions of these routines). NOTE: No output line may exceed 125 (DEC) characters, including the carriage-return.

ROUTINE: PSYSMSG

FUNCTION: Send a line of core image data to the 360

CALLING SEQUENCE: ACC = Number of PDP8 words to be sent

PARAM1 = LOG-1 of the data

PARAM2 = CDF NN specifying the bank of the data

DESCRIPTION:

The routine assumes that a user control character of some sort is in the first word of UBUFF. Each PDP8 word in the block to be sent is converted to two characters in the coded data format (c.f. description of data formats) and is placed into UBUFF following the control character. Then a carriage-return is added, and the line is sent to the 360 in the same manner--and with the same error control--as with ASYSMSG.

NOTE: Since no output line may exceed 125 (DEC) characters, the macimum number of words in the block is 61 (DEC).

ROUTINE: MSGES (not user callable)

FUNCTION: Receive system messages and put into SBUFF

CALLING SEQUENCE: ACC = Last input character

DESCRIPTION:

The routine sets the "G01" UIMON8 system flag immediately (thus inhibiting system routines from sending data to the 630), and sets the "SYSFLG" when the entire message has been received. The input message will be printed on the current SBUFF printer when "UPRIN" (the system buffer controller--not user callable) or SYSERR (user callable) are called.

ROUTINE: LOAD

FUNCTION: Insert loader - formatted data from the 360 into PDP8 core

CALLING SEQUENCE: ACC = Starting address for data

PARAM = CDF NN specifying the bank

DESCRIPTION:

The routine assumes that the input data is in loader format (c.f. data formats); hence, each input character is 6 bits of data or 6 bits of location information. The routine continues loading data until it receives an end-of-data character (211 octal). Since the input data can be of any amount and require more than one input line, carriage-returns and line feeds are ignored until the EOD character is received. The routine can be called explicitly by a user's program in anticipation of a data transmission from the 360. However, if the input is properly formatted, this routine will be automatically called by GMON8 (the remote-input filter routine). With the automatic call feature, and with location information capabilities in the input stream, this routine provides an easy method for automatic data input, display file updating, switch setting, remote program loading, etc.

NOTE: This routine is really a load-and-go routine with a few special switches and options. When the EOD character is received, the routine checks the contents of its core address pointer. If the pointer is not zero, the routine uses that information plus the current bank information as the starting address of a program and branches to it. If the pointer were zero, the routine simply exists to the program that called it. Hence, all blocks of data that are sent through the loader must have the final three characters before

the EOD character specifying new location information. This new location must either be the starting address of the program in the case of a program load, or must be zero in the case of a simple data load.

ROUTINE: SYSMSG

FUNCTION: Communicate fixed messages to the user via the teletype and format output lines from the user's responses.

CALLING SEQUENCE: ACC = N where N is a number > = Ø specifying a particular message residing in a message table defined by the user at system initialization.

DESCRIPTION:

The routine assumes that a message table exists beginning at location 5000 in bank 3, i.e. at the beginning of the user's area. This table is placed into core by the user, either by a call to move the table from the user's program in another bank, by a call to dectape, or by a call to the disk routine. If a response is expected, the routine places the resulting teletype characters into UBUFF beginning at a location specified by the message table. Hence, by carefully constructing his table, the user can automatically format composite output lines from a particular sequence of related inquiries.

The table is formatted as follows:

```
SYSTBL -N              Expected return count.
       UBUFF+M         ADDR of return characters in UBUFF.
       MESSØ-1         ADDR-1 of message.
       ...
       MESSØ           (Characters in packed format)
       MESS1
       ...
       MESSK
```

The expected return count includes a carriage-return that the user will type to end his response. So, if a response of nine characters is expected, -12 would be specified. If the response that the user gave was only six characters instead of nine, the remaining character spaces in the UBUFF buffer would be blanked, and then the carriage-return would be inserted into UBUFF at the end of the nine character field. If the user tried to type more than nine characters, the message would be retyped. If the user typed a percent sign, the message would also be retyped. If the expected return count is specified in the table as -1, no response is assumed, and the result is that only a message is sent to the teletype with no effect on UBUFF.

UBUFF+M (M> = Ø) specifies where in UBUFF the response of the user is to be put. NOTE: UBUFF can hold a maximum of 125 (DEC) characters including carriage-return.

MESSØ-1 specifies the starting location of the message to be printed on the teletype.

-L specifies the length in characters of the message. If the expected return count were -1, the system adds a line feed and carriage-return to the line printed.

The table (K entries of four words each) is followed by the K variable length messages.

NOTE: The user could specify some other buffer area in bank 3 as the recipient of the teletype responses; however, if the user does this, it is his responsibility to insure that no part of the UIMON8 system is damaged.

ROUTINE: SYSERR

FUNCTION: Test the system buffers and return a code to the caller

CALLING SEQUDNCE: None (ACC assumed = ∅)

DESCRIPTION:

It is possible that the 360 or the PDP7 will at sometime send a message to the PDP8 which is unrelated to the user's program, i.e. time sharing off, et. al. If UIMON8 is in time sharing or text editor mode at the time, these messages would automatically be printed on the teletype. However, in other modes (either system or user defined), immediate print-out would not necessarily occur, since printing is only done when a call to the system print routine is initiated. The complete arrival of a system message is indicated when UIMON8 sets its "SYSFLG" to nonzero. When called, this routine prints the system buffers, and then tests "SYSFLG"; if the flag were up (nonzero), the routine prints "CODE:" and waits for a teletype reply. This reply (a number zero through seven) is returned to the calling program in the accumulator, and the "SYSFLG" is reset to zero. Hence, the calling program has the option of taking various courses of action for any particular system message.

ROUTINE: MOVEDT

FUNCTION: Move a block of PDP8 core to anywhere in the machine

CALLING SEQUENCE: ACC = FØTØ (F=BANK FROM, T=BANK TO)

PARAM1 = ADDR FROM

PARAM2 = ADDR TO

PARAM3 = -COUNT (in words)

DESCRIPTION:

The data at the specified "FROM" address in the "F" bank is moved sequentially starting at the lowest core address, and is moved to the "TO" address in the "T" bank. The user is cautioned about overlapping fields and should note that the action of this routine is identical to a 360 MVC instruction. This is merely a utility type routine and is included as a convenience for all users.

APPENDIX C

Features of 360/50-PDP7-PDP8 Communications

1. PDP8 to PDP7 to 360/50

The PDP8 transmits or receives one character at a time to or from the PDP7 at a maximum rate of 100 characters per second. As soon as the PDP7 senses a carriage-return character from the PDP8 or when it has received a total of 125 characters without getting a carriage-return character, the PDP7 considers the "line" of characters from the PDP8 to be completed. At this time the PDP7 transmits a carriage-return followed by a line feed to the PDP8. The PDP7 then tries to send the line to the 360/50. When the 360/50 reads the line, and not before, the PDP7 sends a bell-character to the PDP8 signifying that the PDP8 may send another line. If the PDP8 should try to send characters to the PDP7 before receiving the bell-character, the PDP7 immediately sends a carriage-return line feed, "#WAIT", carriage-return, line feed sequence to the PDP8.

The format of the line sent to the 360/50 from the PDP7 is as follows:

```
A,B[125 data bytes]CR
0 1 2          126 127
```

Where A and B (bytes 0 and 1) are PDP7-360/50 control characters, byte 127 is always a carriage-return, and bytes 2 through 126 are data bytes. If the PDP8 had sent 50 characters followed by a carriage-return, the line would look like:

```
A,B[.....CR---Junk---]CR
0 1 2   51 52       126 127
```

2. 360/50 to PDP7 to PDP8

When the 360/50 sends data for the PDP8 to the PDP7, the data format is:

```
AB[N---124 data bytes]C R
01 23              126 127
```

where A and B are PDP7-360/50 control characters, N is a control character for the PDP8 monitor system, and byte 127 is always a carriage-return. If 50 characters were to be sent to the PDP8, the line would be:

```
AB[N      CR---Junk---]C R
01 23 52 53 54      126 127
```

The PDP7 would send only bytes 2 through 53 to the PDP8, followed by an additional line feed (NOTE: When the PDP8 initiated a send, a line feed and a bell were returned; when the /50 initiates a send, only a line feed is additionally sent to the PDP8.).

After sending one line, the /50 cannot send another line until the first has been completely sent (character by character) to the PDP8. Note that the PDP7 assumes the PDP8 is a synchronous device; hence, the PDP7 sends characters to the PDP8 at a fixed rate and the PDP8 must accept them at that rate. In order to know when this has been accomplished, the program in the /50 must issue a READ operation. In this case, byte 1 ('B') of the input indicates the disposition of the last line. (On ordinary input, B = 0, meaning the line is a data line.) The rest of the line is junk. If B is

1, then the output has all been transmitted to the PDP8.

or 2, then the PDP8 has requested the PDP7 to stop transmission before the entire line was sent, i.e. the remainder of the line has been voluntarily lost;

or 3, then the PDP8 has "signed off."

Hence, to send several lines to the PDP8, the program in the /50 must issue a read before every output operation (except the first) and check B for the proper setting.

3. Data Format

Since certain characters, such as carriage-return and line feed cause the PDP7 to take some action, data--particularly object code--cannot be sent 8 bits at a time from the PDP8; nor is 8 bits really convenient considering the 12 bit word of the PDP8. In addition, the 360/50 cannot send data in its original form to the PDP7 either, since an inadvertant carriage return in the middle of the line would cause the PDP7 to stop transmitting to the PDP8. Hence, all data (except pure character data sent only to the time-sharing system) is sent in a coded format, six data bits (half of a PDP8 word) per 8 bit character at a time. Since internal PDP8 characters are 6 bits and the transmitted form takes up 8 bits (1 byte), conversion in the 360/60 to EBCDIC is very neat and clean. Numeric data is reformed into 360/50 half words, i.e. one PDP8 word of 12 bits becomes 2 bytes. In the PDP8 each incoming data byte is stripped of the excess 2 bits and every 2 characters are packed into one PDP8 word.

APPENDIX D

PDP8 Data Formats

The UIMON8 system works with three types of data--ASCII, packed ASCII, and Loader formatted.

ASCII characters are stored one character per word in the right-most 8 bits. This data is always of a temporary nature, residing only in buffers just prior to being sent to the teletype or to the 360, or just after having been received from those devices.

Packed ASCII (really 6 bit trimmed ASCII corresponding identically to the right-most 6 bits of regular 8 bit ASCII characters) is used in the text editor display buffer, in all stored system messages, and is assumed to be the format of user messages stored in the user's area for the SYSMSG facility. When one of these messages is to be sent to the teletype, a system routine unpacks this data into UBUFF. Since the code is 6 bits, there are 64 valid print characters available in stored messages or for display in the text editor, except that three of these are control characters for the text editor. These three are line feed (33), carriage-return (34), and escape data state (35), all of which should not be used by the user.

Loader formatted data is used to pass arbitrary bit patterns into and out of the PDP8. Since certain characters are control characters for the PDP7/360 system (such as carriage-return), UIMON8 must insure that such characters are not sent accidentally during the transmission of a data block. Hence, each PDP8 word of data is converted into two characters before being sent to the 360. Each character is of the form 1(6DATA BITS)0, with the top six bits of the PDP8 word going into the first character, and the right-most six going into the second

character. Naturally, data coming into the system by way of the load routine is of the same form. The only other characters in this type are carriage-returns (215 octal), line feeds (212), EOD (211), and origin characters (20X, where X is 1, 3, 5, or 7 specifying banks 0 to 3, respectively). The only ambiguity in this setup is with line feed which could be mistaken for a data character. However, line feeds are only sent by the PDP7 after a carriage-return at the end of each line, so by definition, 212 is only a line feed if it occurs after a carriage-return. Whenever an origin character occurs, the bank information is extracted and used as the current load bank for the input data, and the next two data characters are assumed to contain a new starting address in that bank instead of being data.

APPENDIX E

UIMON8 Interrupt Tables

As explained in Section IV. B. 2, PDP8 routines, the first table contains test IOT's for interrupts to be investigated and clear IOT's for interrupts to be ignored. The second table contains the address of the routine to be entered if the corresponding test IOT in the first table is successful.

Both TORCL and INTLST appear as follows at UIMON8 initialization, and the system assumes that the devices are in this order.

```
TORCL 6031          TEST KEYBOARD FLAG
      6041          TEST TELEPRINTER FLAG
      6631          TEST 630 REMOTE FLAG
      6132          TEST LIGHT PEN HIT FLAG
      7000          CLEAR MANUAL INTERRUPT FLAG
      7000          CLEAR EDGE FLAG
      7000          CLEAR EXTERNAL FLAG
      7000          CLEAR INTERNAL INTERRUPT
      6314          CLEAR CLOCK INTERRUPT
      6764          CLEAR DECTAPE FLAG
      SZA           TEST FOR PUSH BUTTON HIT
INTLST KEYTS        KEYBOARD
      TPRTR         TELEPRINTER
      GMON8         REMOTE FILTER
      LPEN          LIGHT PEN
      IRTRN         NO-OP
      IRTRN         NO-OP
      IRTRN         NO-OP
      IRTRN         NO-OP
      IRTRN         NO-OP
      IRTRN         NO-OP
      PBHIT?        PUSH BUTTON HIT
      PBHITS        IF PUSH BUTTON HIT, NO-OP
```

NOTE: If none of the other devices caused an interrupt, PBHIT? is entered to clear the flag; then PBHIT? calls PBHITS if the flag was on. Hence, user's should change PBHITS and not PBHIT?.

APPENDIX F

UIMON8 System Locations

(All locations in bank 3)

SYSTEM START ("UTIL")---1000

ASYSMSG----------1600

PSYSMSG----------1614

LOAD----------1464

SYSMSG----------1656

SYSERR----------1731

MOVEDT----------3674

UBUFF----------3030

SBUFF----------3230

SYSFLG----------0007

The following routines may only be called by programs in bank 3, in the user's area, for example.

SCSET----------1211 (TS CONSOLE)

LD360----------1235 (LOAD 360 PARTITION)

LDPDP8----------1261 (LOAD FROM 360 INTO PDP8)

TEXTED----------1306 (TEXT EDITOR)

TERM8----------1327 (UIMON8 EXIT TO DECTAPE)

SYSTEM INTERRUPT TABLES:

TORCL----------1400 (TEST OR CLEAR IOT TABLE)

INTLST----------1413 (INTERRUPT ROUTINE LIST)

APPENDIX G

Character Code Equivalences

STANDARD (EBCDIC)	TYPE A (ASCII)	TYPE B (CODED ASCII)	TYPE C (CODED BCD)
A	C1	82	E2
B	C2	84	E4
C	C3	86	E6
D	C4	88	E8
E	C5	8A	EA
F	C6	8C	EC
G	C7	8E	EF
H	C8	90	F0
I	C9	92	F2
J	CA	94	C2
K	CB	96	C4
L	CC	98	C6
M	CD	9A	C8
N	CE	9C	CA
O	CF	9E	CC
P	D0	A0	CE
Q	D1	A2	D0
R	D2	A4	D2
S	D3	A6	A4
T	D4	A8	A6
U	D5	AA	A8
V	D6	AC	AA
W	D7	AE	AC
X	D8	B0	AE
Y	D9	B2	B0
Z	DA	B4	B2
0	B0	E0	94
1	B1	E2	82
2	B2	E4	84
3	B3	E6	86
4	B4	E8	88
5	B5	EA	8A
6	B6	EC	8C
7	B7	EE	8E
8	B8	F0	90
9	B9	F2	92
NOT	87 BELL	87 BELL	FA BELL
UNDERSCORE	8A LF	UNUSED	FC LF
CENTS	89 EOD	89 EOD	89 EOD
PERCENT	A5	CA	FE
.	AE	DC	F6
<	BC	F8	BC
(	A8	D0	B8
+	AB	D6	E0
UPARROW	DE	BC	A0
AMPERSAND	A6	CC	80
!	A1	C2	9A
$	A4	C8	D6
*	AA	D4	D8

)	A9	D2	F8
;	BB	F6	BA
-	AD	DA	C0
/	AF	DE	A2
,	AC	D8	B6
>	BE	FC	BE
?	BF	FE	D4
:	BA	F4	9E
NUMSN	A3	C6	B4
EACH	C0	80	DA
'	A7	CE	98
=	BD	FA	96
"	A2	C4	9C
BLANK	A0	C0	80

ALL TYPES A, B, AND C ARE IN HEX (I.E. A = 1010 BINARY)

APPENDIX H

JCL Examples

1. Basic JCL for assembling 360 programs with communications macros:

```
/*ID
//  EXEC ASM
//ASM.SYSLIB DD DSN=SYS1.MACLIB,DISP=OLD
//  DD DSN=SYS3.MACLIB,DISP=OLD
//  DD DSN=USER.GRAPHICS,MACLIB,DISP=OLD,UNIT=2314,    X
//          VOLUME=SER=UIRUR1
//ASM.SYSIN DD *
```

SOURCE DECK

```
/*
```

2. To add the object code of properly assembled programs to the system, change the EXEC ASMLKED card and, after the /*, add:

```
//LKED.SYSLMOD DD DSN=USER.GRAPHICS.LINKLIB(YOURNAME),    X
//         UNIT=2314,DISP=OLD,VOLUME=SER=UIRUR1
```

3. Basic 360 JCL for assembling PDP8 programs with PDP8ASM:

```
/*ID
//JOBLIB DD UNIT=2314,VOLUME=SER=UIRUR1,    X
//  EXEC PGM=PDP8ASM
//SYSUT1 DD UNIT=DRUM,SPACE=(TRK,(10,10)),DISP=NEW
//SYSUT8 DD DUMMY
//SYSUDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
```

SOURCE DECK

```
/*
```

4. To put the object code into the monitor system, change the SYSUT8 card to read:

```
//SYSUT8 DD DSN=USER.GRAPHICS.LINKLIB(YOURNAME),    X
//          DISP=OLD,UNIT=2314,VOLUME=SER=UIRUR1
```

5. To punch the object code on a paper tape on the PDP7 in loader format, change the SYSUT8 card to:

```
//SYSUT8 DD UNIT=(CTC,,DEFER)
```

and add the following ASP control cards before the JOBLIB card:

```
/*PROCESS MAIN
/*PROCESS FILE
/*FORMAL FI,DD=SYSUT8,DEST=PUNCH7,CONTROL=SINGLE
/*PROCESS PRINT
/*ENDPROCESS
```

APPENDIX I

GRASS Acronyms

Components of GRASS software support are:

DOOFIS: Disk Oriented Operating and FIling System--for the satellite PDP8.

FLOG: Facility for Local Online Graphics--individual graphics terminal support running under DOOFIS and GLASP on the PDP8.

GADS: Graphical Applications Drawing System--for 360/75 programs to create displays for the graphics terminals of the PDP8.

GASP: Graphics Attached Support Processor--communications and monitor system for the graphics region of the 360/75.

GEAR: Graphically Extended Analysis Routine--any analysis package running under GASP.

GIRLS: Graphical-Information Retrieval and Library System--an IR facility with sections resident and active in both CPU's.

GLASP: Graphics Locally Attached Support Processor--PDP8 complement of GASP.

SYMP: SYmbolic Manipulation Programs--interface and pre-processor for GEARs.

m AEC-427
(6/68)
ECM 3201

U.S. ATOMIC ENERGY COMMISSION
UNIVERSITY-TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

AEC REPORT NO.	2. TITLE
COO-1469-0144	GRAPHICAL REMOTE-ACCESS SIMULATION SYSTEM (GRASS) The Communication and Monitor Components (GASP/GLASP)

TYPE OF DOCUMENT (Check one):

- [X] a. Scientific and technical report
- [ ] b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- [ ] c. Other (Specify) _____

RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- [X] a. AEC's normal announcement and distribution procedures may be followed.
- [ ] b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- [ ] c. Make no announcement or distrubution.

REASON FOR RECOMMENDED RESTRICTIONS:

SUBMITTED BY: NAME AND POSITION (Please print or type)

C. W. Gear, Professor and
Principle Investigator

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature	Date
Charles W Gear.	August 1969

FOR AEC USE ONLY

AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

PATENT CLEARANCE:

- [ ] a. AEC patent clearance has been granted by responsible AEC patent group.
- [ ] b. Report has been sent to responsible AEC patent group for clearance.
- [ ] c. Patent clearance not required.